Population based algorithms: Evolutionary algorithms & Swarm intelligence

Inspired from

Metaheuristics: From Design to Implementation, Chapter 3, Talbi, El-Ghazali

Dr. Gwen Maudet

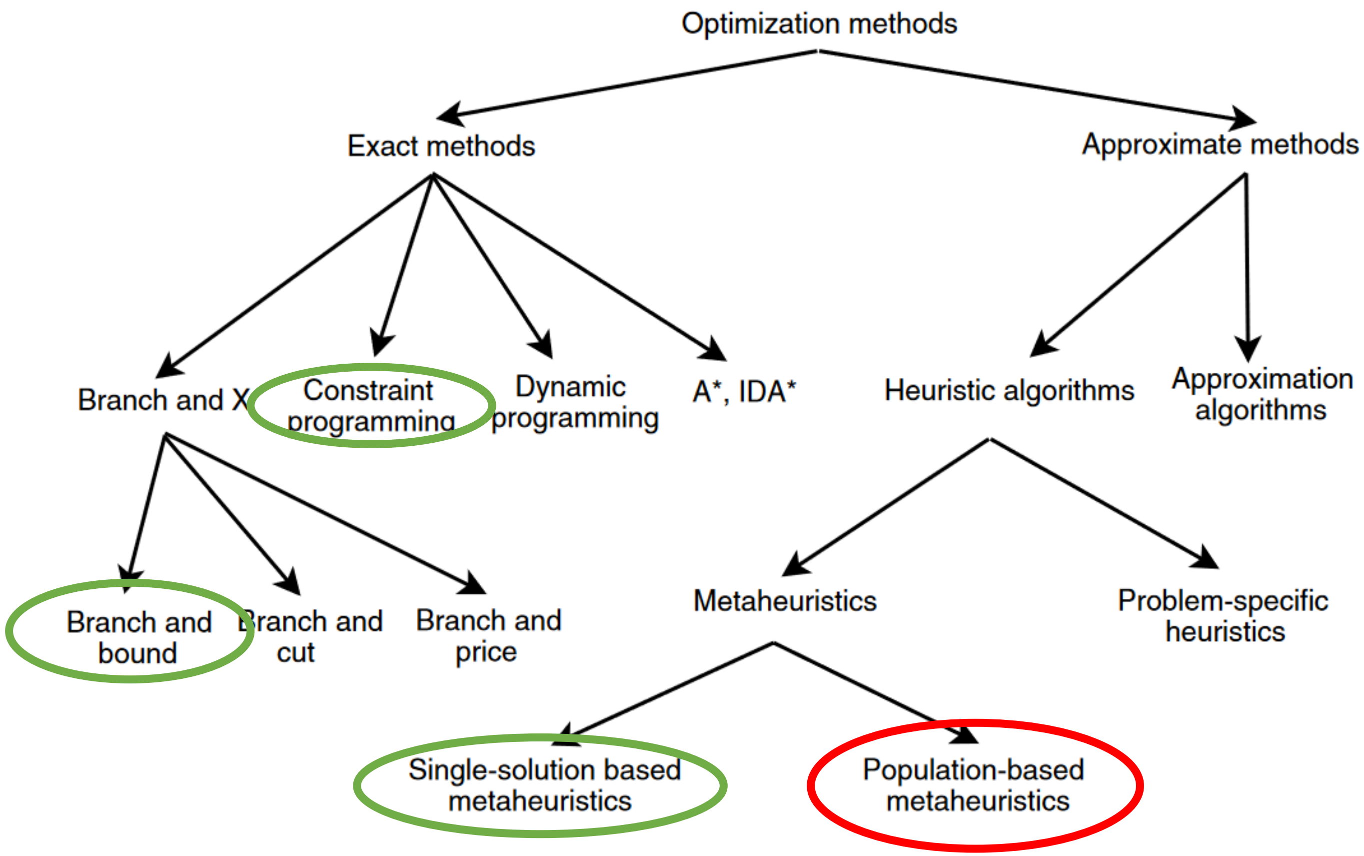
december 2025

UNIVERSITÉ DU LUXEMBOURG

Where are we

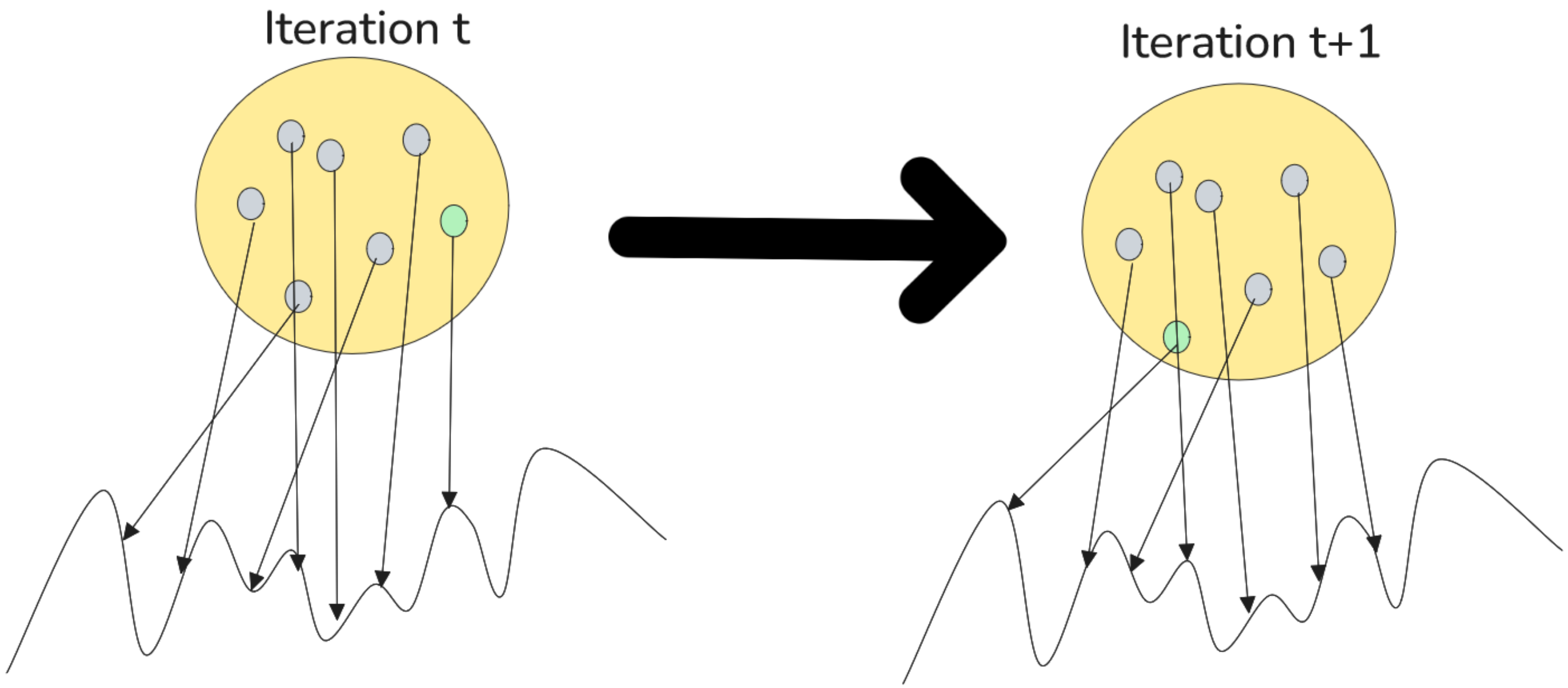
- 19/09 : Introduction (Greg)
- 23/09 : Problem modelling (Greg)
- 30/09 : Graphic solving (Greg)
- 07/10 : Branch & Bound (Gwen)
- 14/10 : Branch & Bound 2 (Gwen)
- 21/10 : Consistency and All Different Global Constraint (Pierre)
- 28/10 : Scheduling Problem and Cumulative Global Constraint (Pierre)
- 04/11 : Implementation of Propagate-and-Search in Python (Pierre)
- 11/11 : Multiobjective Optimization (Greg)
- 18/11 : Heuristic algorithms : local search (Greg)
- 25/11 : approximation algorithms: population based (Gwen)
- 02/12 : Put all this in practice in a jupiter notebook (Gwen)
- 09/12 : Put all this in practice in a jupiter notebook II (Gwen)
- 16/12 : Ongoing Research in Optimisation (Gwen + Greg)

Where are we



Optimization methods
Exact methods
Approximate methods
Branch and X
Constraint programming
Dynamic programming
A*, IDA*
Heuristic algorithms
Approximation algorithms
Branch and bound
Branch and cut
Branch and price
Metaheuristics
Problem-specific heuristics
Single-solution based metaheuristics
Population-based metaheuristics

Population based algorithms



Main framework of population based algo

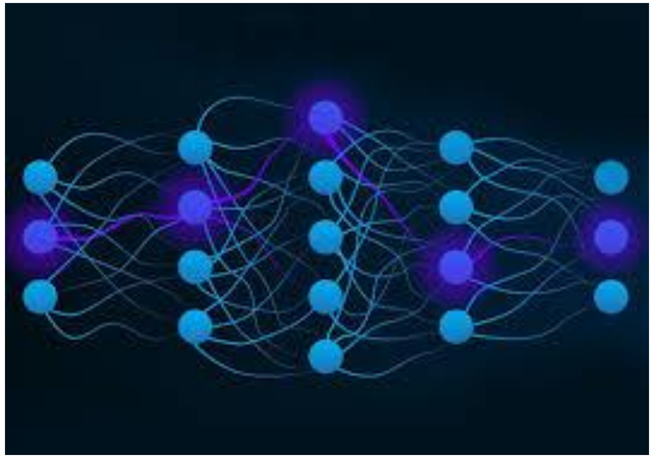
Algorithm 3.1 High-level template of P-metaheuristics.

$P = P_0$; /* Generation of the initial population */
$t = 0$;
Repeat
  Generate(P'_t); /* Generation a new population */
  P_{t+1} = Select-Population($P_t \cup P'_t$); /* Select new population */
  $t = t + 1$;
Until Stopping criteria satisfied
Output: Best solution(s) found.

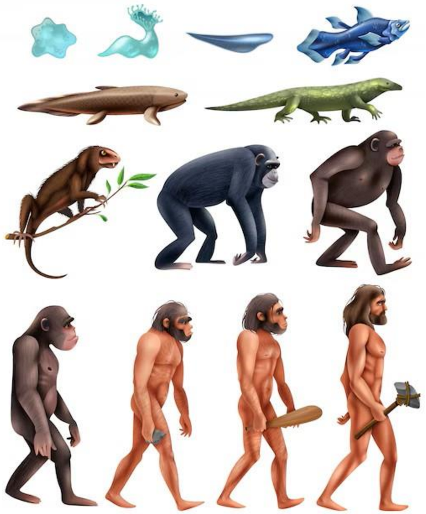
Outline

- Common concepts on Population based metaheuristics
- **Evolutionary algorithms**
- **Swarm intelligence**

Nature–inspired algorithms



- Simulated annealing
- Tabu search
- Quantum computing
- Neural networks
- …
- **Evolutionary Algorithms**
- **Swarm intelligence**



Outline

- Common concept on Population based metaheuristics
- **Evolutionary algorithms**
- **Swarm intelligence**

Overview of existing initialization methods

Strategy	Diversity	Computational Cost	Quality of Initial Solutions
Pseudo-random	++	+++	+
Quasi-random	+++	+++	+
Sequential diversification	++++	++	+
Parallel diversification	++++	+++	+
Heuristic	+	+	+++

Pseudo-random VS Parallel diversification

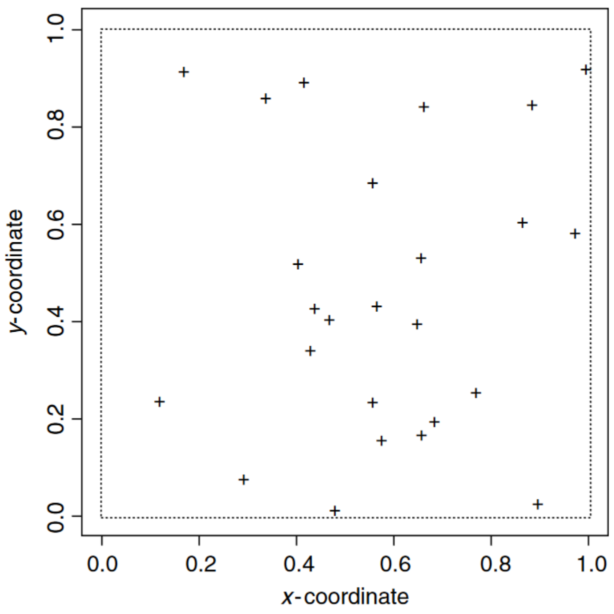


FIGURE 3.4 In the pseudo-random generation, 25 solutions are generated independently in the search space.

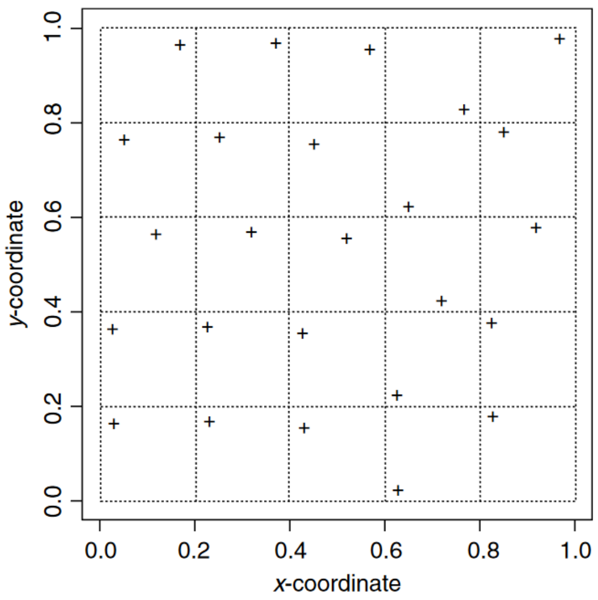


FIGURE 3.3 In the Latin hypercube strategy, the search space is decomposed into 25 blocks and a solution is generated pseudo-randomly in each block.

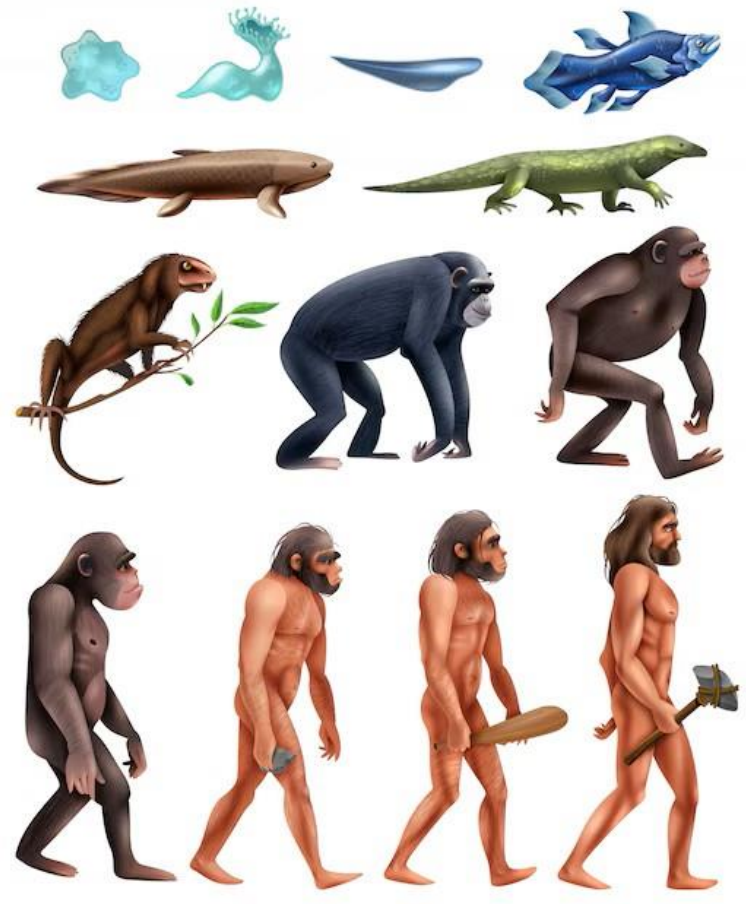
Stopping criteria

- Static procedure
  - Number of iteration
  - Computation time
  - …
- Adaptive procedure
  - Number of iterations without improvements
  - Diversity of the population
  - Optimal or satisfactory solution is reached

Outline

- Common concepts on Population based metaheuristics
- **Evolutionary algorithms**
- **Swarm intelligence**

Principle of evolution



- Evolution through mutations, crossovers for each generations

→Best offsprings are kept for next generations

TABLE 3.3 Evolution Process Versus Solving an Optimization Problem

Metaphor	Optimization
Evolution	Problem solving
Individual	Solution
Fitness	Objective function
Environment	Optimization problem

A bit of history

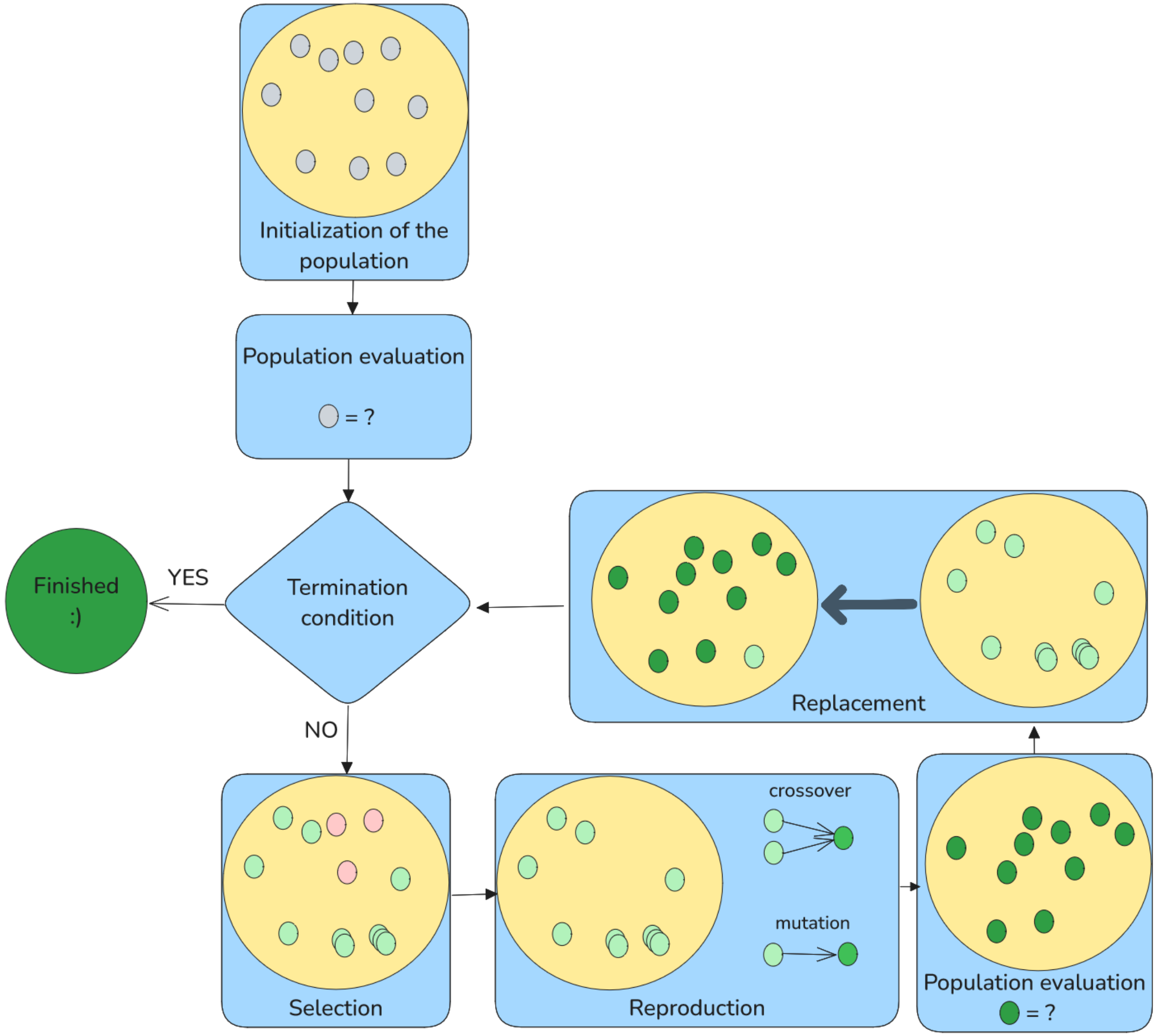
- **Evolutionary Programming**: L. Fogel (1962)

- **Genetic Algorithms**: J. Holland (1962)

- **Evolution Strategies:** I. Rechenberg & H.-P. Schwefel (1965)

- **Genetic Programming**: J. Koza (1989)

Evolutionary algorithms



Domains of application

- Numerical, Combinatorial Optimisation
- System Modeling
- Planning and Control
- Data Mining
- Machine Learning

…

Performance

- Acceptable performance at acceptable costs on a wide range of problems
- Intrinsic parallelism (robustness, fault tolerance)
- Superior to other techniques on complex problems with
  - Lots of data, many free parameters
  - Complex relationships between parameters
  - Many (local) optima
  - Adaptive, dynamic problems

Advantages

- No presumptions w.r.t. problem space
- Widely applicable
- Low development & application costs
- Easy to incorporate other methods (hybridization)
- Solutions are interpretable (unlike NN)
- Provide many alternative solutions
- Robust regards any change of the environment (data, objectives, etc)

disadvantages

- No guarantee for optimal solution within finite time (in general)
- May need parameter tuning
- Often computationally expensive, i.e. slow (when fitness evaluation is expensive)

Components of an EA

- Representation of an individual

- Initialization method
- Objective function
- Selection strategy
- Reproduction strategy
- Replacement strategy
- Termination criterion

Algorithm 3.2 Template of an evolutionary algorithm.

Generate($P(0)$) ; /* Initial population */
$t = 0$;
While not Termination_Criterion($P(t)$) **Do**
  Evaluate($P(t)$) ;
  $P'(t)$ = Selection($P(t)$) ;
  $P'(t)$ = Reproduction($P'(t)$); Evaluate($P'(t)$) ;
  $P(t+1)$ = Replace($P(t)$, $P'(t)$) ;
  $t = t + 1$;
End While
Output Best individual or best population found.

Components of an EA

- Representation of an individual

- Initialization method
- Objective function
- Selection strategy
- Reproduction strategy
- Replacement strategy
- Termination criterion

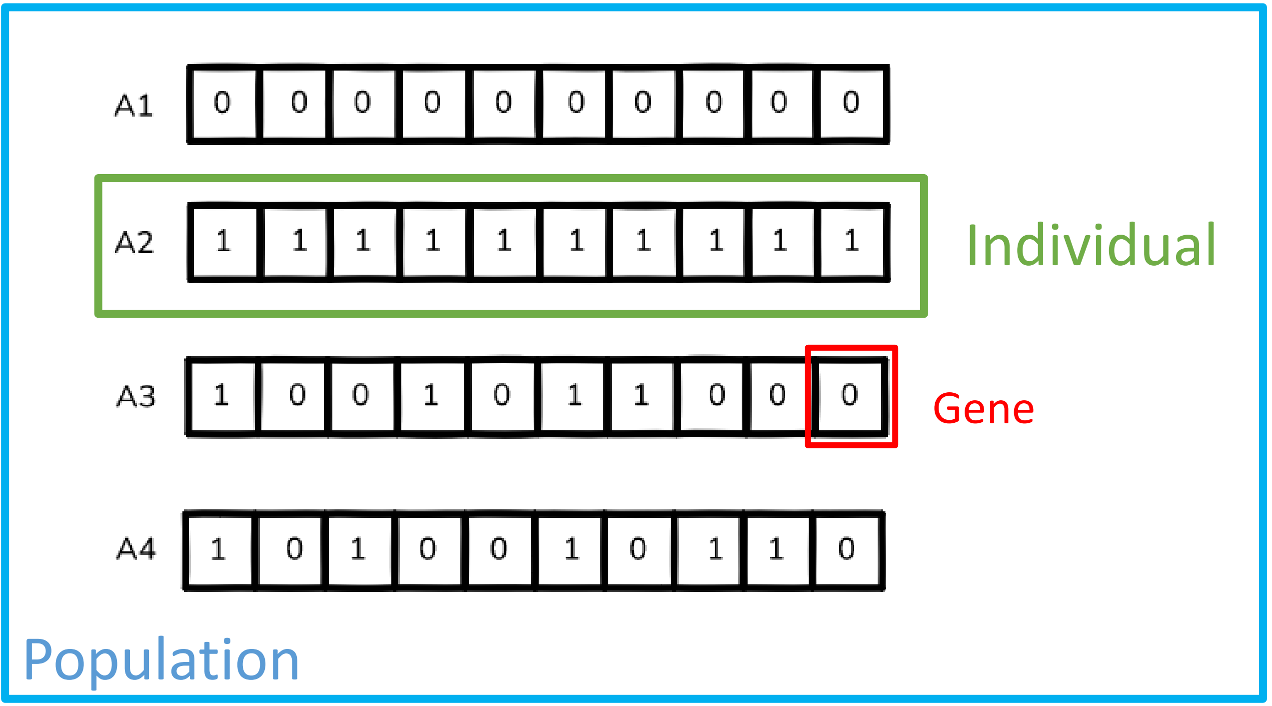
Algorithm 3.2 Template of an evolutionary algorithm.

```
Generate(P(0)) ; /* Initial population */
t = 0 ;
While not Termination_Criterion(P(t)) Do
    Evaluate(P(t)) ;
    P'(t)      = Selection(P(t)) ;
    P'(t)      = Reproduction(P'(t)); Evaluate(P'(t)) ;
    P(t + 1)   = Replace(P(t), P'(t)) ;
    t = t + 1 ;
End While
Output Best individual or best population found.
```

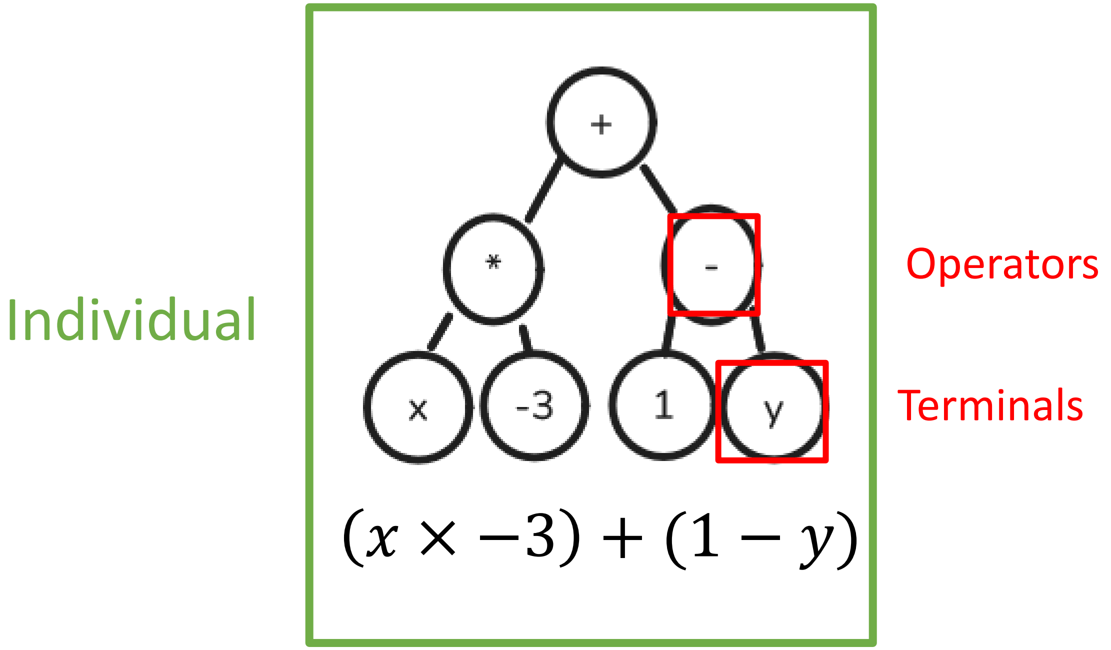
Types of EA representations

- **Genetic Algorithm**: one individual is a list

Used in discrete optimisation



- **Genetic Programming**: one individual is a program



- **Evolution strategies, Evolutionary programming, Differential evolution..**

Components of an EA

- Representation of an individual

- Initialization method
- Objective function
- Selection strategy
- Reproduction strategy
- Replacement strategy
- Termination criterion

Algorithm 3.2 Template of an evolutionary algorithm.

Generate($P(0)$) ; /* Initial population */
$t = 0$;
While not Termination_Criterion($P(t)$) **Do**
  Evaluate($P(t)$) ;
  $P'(t)$ = Selection($P(t)$) ;
  $P'(t)$ = Reproduction($P'(t)$); Evaluate($P'(t)$) ;
  $P(t+1)$ = Replace($P(t)$, $P'(t)$) ;
  $t = t + 1$;
End While
Output Best individual or best population found.

Objective function

- Quantify the quality of an individual

SUPER IMPORTANT: represent the desired traits of an individual; discriminating factor during selection.

Components of an EA

- Representation of an individual

- Initialization method
- Objective function
- Selection strategy
- Reproduction strategy
- Replacement strategy
- Termination criterion

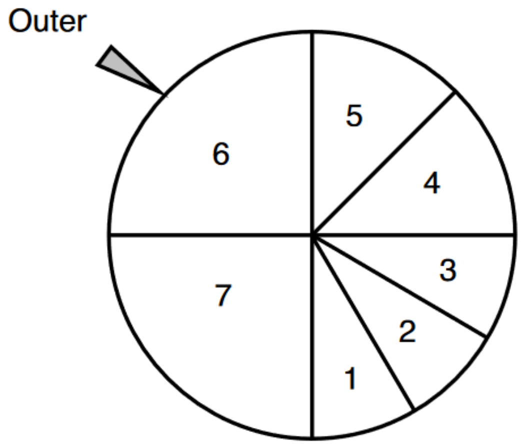
Algorithm 3.2 Template of an evolutionary algorithm.

Generate($P(0)$) ; /* Initial population */
$t = 0$;
While not Termination_Criterion($P(t)$) **Do**
  Evaluate($P(t)$) ;
  $P'(t)$ = Selection($P(t)$) ;
  $P'(t)$ = Reproduction($P'(t)$); Evaluate($P'(t)$) ;
  $P(t+1)$ = Replace($P(t)$, $P'(t)$) ;
  $t = t + 1$;
End While
Output Best individual or best population found.

Selection strategy

Individuals:	1	2	3	4	5	6	7
Fitness:	1	1	1	1.5	1.5	3	3

- roulette



- Tournament

Size, e.g. k=3:

i VS j VS k → best fitness(i, j, k)

- Stochastic universal sampling, Rank based selection

Components of an EA

- Representation of an individual
- Initialization method
- Objective function
- Selection strategy
- Reproduction strategy
- Replacement strategy
- Termination criterion

Algorithm 3.2 Template of an evolutionary algorithm.

Generate($P(0)$) ; /* Initial population */
$t = 0$;
While not Termination_Criterion($P(t)$) **Do**
  Evaluate($P(t)$) ;
  $P'(t)$ = Selection($P(t)$) ;
  $P'(t)$ = Reproduction($P'(t)$); Evaluate($P'(t)$) ;
  $P(t+1)$ = Replace($P(t)$, $P'(t)$) ;
  $t = t + 1$;
End While
Output Best individual or best population found.

Reproduction strategy

Depend highly on the representation of an individual

- **Mutation**: which modifies an individual.

Ergodicity: every solution in the search space should be reached

Locality: minimal change (related to neighborhood)

Valid: provides valid solution

Low probability $0.001 \leq p \leq 0.01$

- **Crossover**: which combines two or more individuals to generate new ones

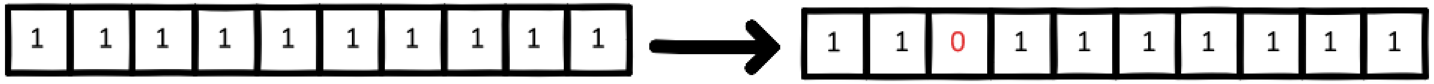
Heritability: should inherit characteristics from both parents

Valid: provide valid solution

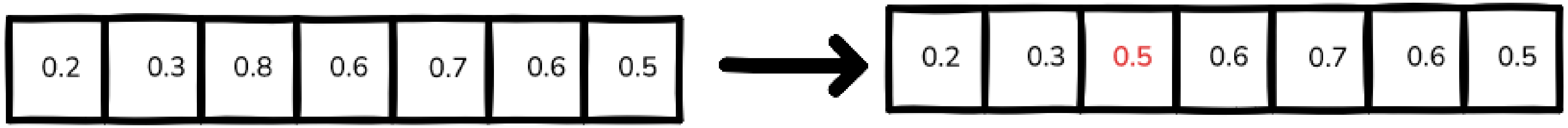
High probability $0.45 \leq p \leq 0.95$

Usual mutations of GA

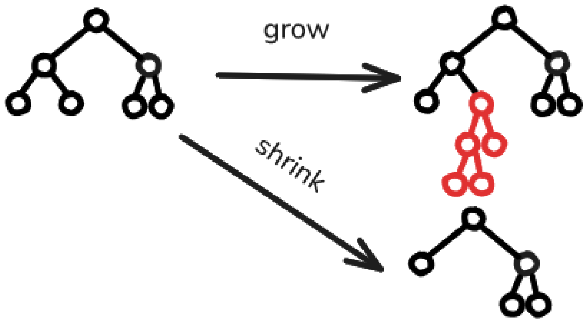
- Binary representation: flip operator.



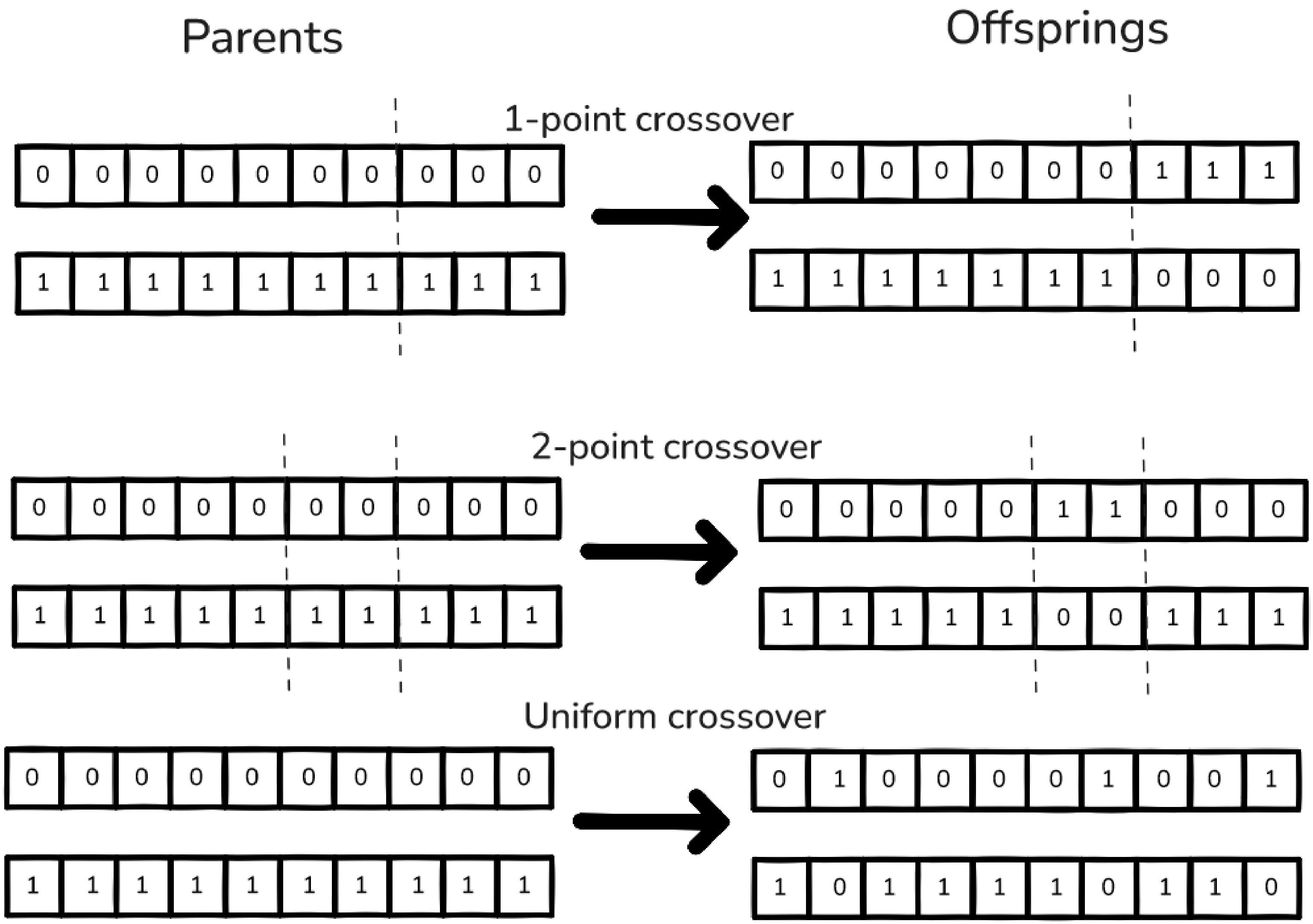
- Discrete representation: changing the value associated with an element by another value: $x_i' = x_i + N(0, \sigma)$ for instance



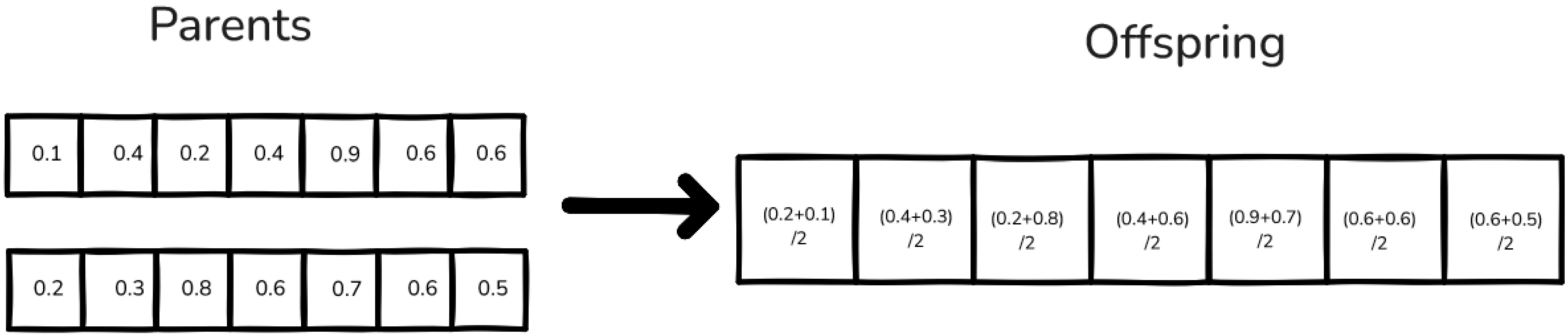
- tree representation: growing, shrink the tree



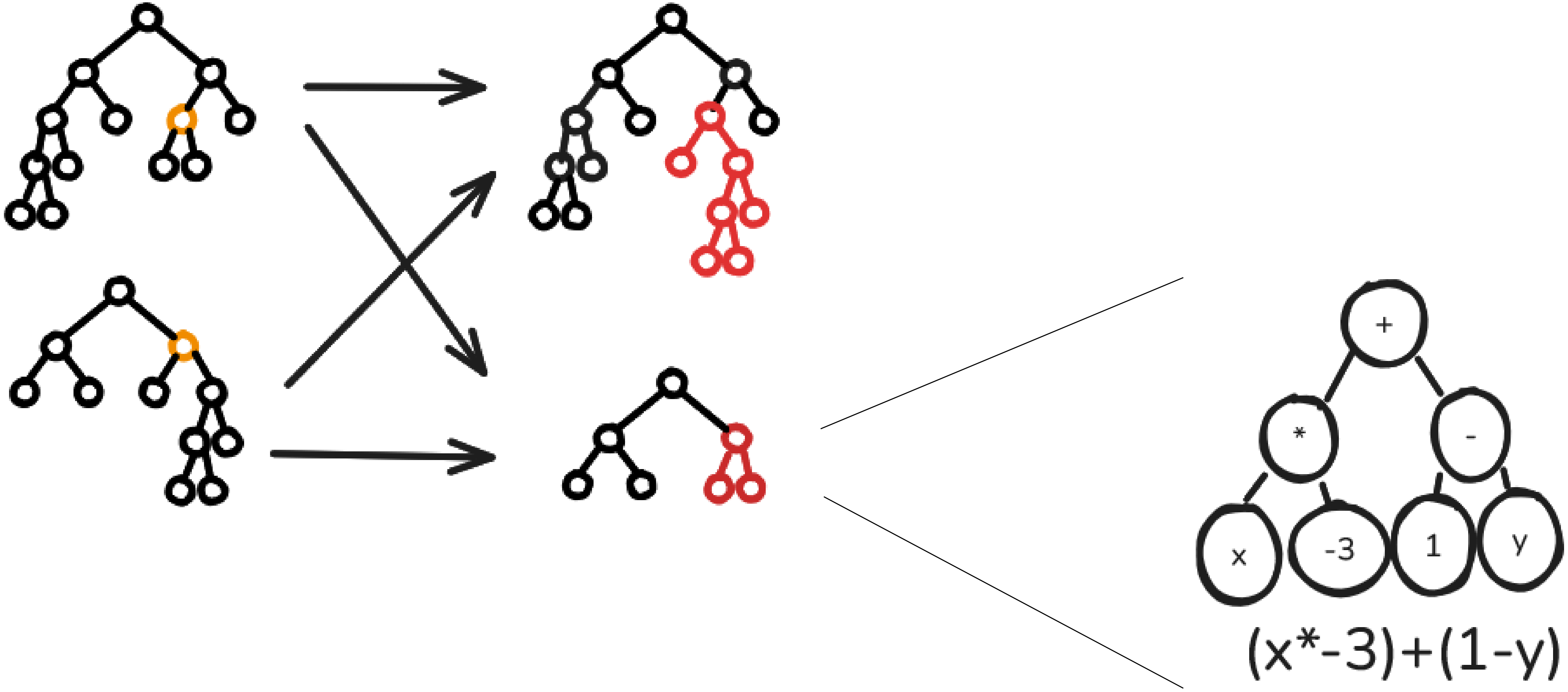
Crossovers in binary representation



Arithmetic crossover for real number



Crossover in tree representation



Components of an EA

- Representation of an individual
- Initialization method
- Objective function
- Selection strategy
- Reproduction strategy
- Replacement strategy
- Termination criterion

Algorithm 3.2 Template of an evolutionary algorithm.

Generate($P(0)$) ; /* Initial population */
$t = 0$;
While not Termination_Criterion($P(t)$) **Do**
  Evaluate($P(t)$) ;
  $P'(t)$ = Selection($P(t)$) ;
  $P'(t)$ = Reproduction($P'(t)$); Evaluate($P'(t)$) ;
  $P(t+1)$ = Replace($P(t)$, $P'(t)$) ;
  $t = t + 1$;
End While
Output Best individual or best population found.

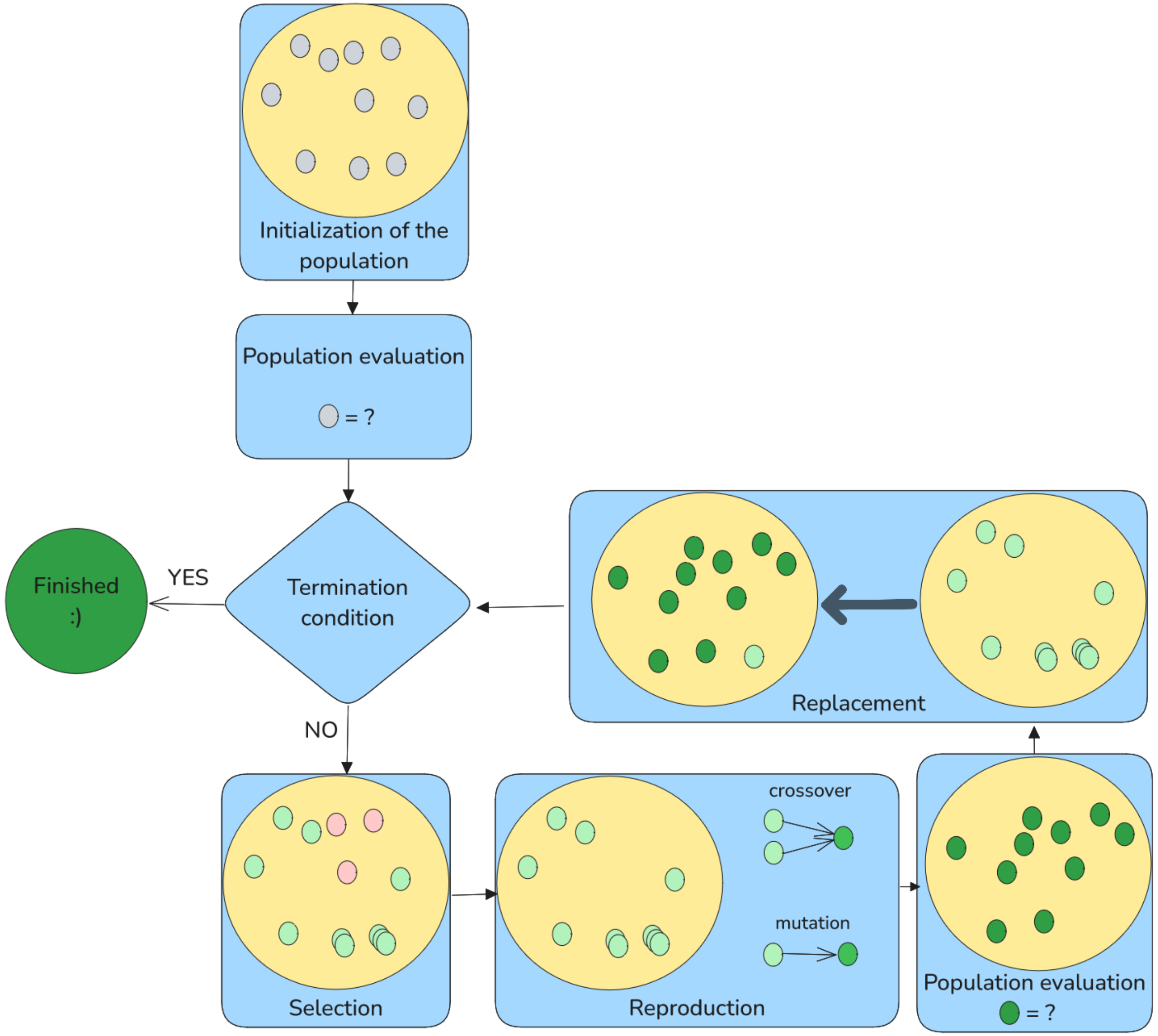
Replacement strategy

Represents the survivor selection of both the parent and the offspring populations.

- **Generational replacement**: The replacement will concern the whole population. The offspring population will replace systematically the parent population.
- **Steady-state replacement**: At each generation of an EA, only one offspring is generated. For instance, it replaces the worst individual of the parent population.

Can include **elitism**: reintroducing the best solution found so far.

Evolutionary algorithms



Also… things to keep in mind

- EAs are stochastics: don’t draw any conclusions from a single run

- EA’s core evolutionary component is about comparison : do fair competitions.
    - The objective function should be in intelligently chosen
    - Offsprings should have a chance to be better than the parents

Some exercices

Small exercise 1

- Suppose a genetic algorithm uses chromosomes of the form $x = abcdefgh$: a fixed length of eight genes with each gene being any digit between 0 and 9. Let the fitness of individual x be calculated as:

$$f(x) = (a + b) - (c + d) + (e + f) - (g + h)$$

- Initial population is

$$x_1 = 65413281$$
$$x_2 = 12342901$$

1. Evaluate the fitness of x_1, x_2.
2. Evaluate the two offspring, considering crossover:
   1. using the one-point crossover at the middle point;
   2. using the two-point crossover at b-c and e-f points.

Small exercise 2

- The knapsack problem is defined by:

The **knapsack problem** is the following problem in combinatorial optimization:

> *Given a set of items, each with a weight and a value, determine which items to include in the collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.*

- Model the problem and define a solution of the problem:
  - How is defined a gene and an individual
  - How is defined the fitness

Small exercise 2 SOLUTION

- Set of items $(x_i)_{1 \leq i \leq n}$; each item x_i has a weight w_i with a value v_i.
- Select $S \subset [1, n]$ the set $(x_i)_{i \in S}$:
  - Maximize $\sum_{i \in S} v_i$
  - such that $\sum_{i \in S} w_i \leq W$ the maximum weight

Small exercise 2 SOLUTION

- The representation of a solution is a binary list, where 1 means that it is included in the bag, 0 otherwise

A1	0	0	0	0	0	0	0	0	0	0
A2	1	1	1	1	1	1	1	1	1	1
A3	1	0	0	1	0	1	1	0	0	0
A4	1	0	1	0	0	1	0	1	1	0

- Fitness is defined by the value of the selected items if it respects the weights, 0 otherwise

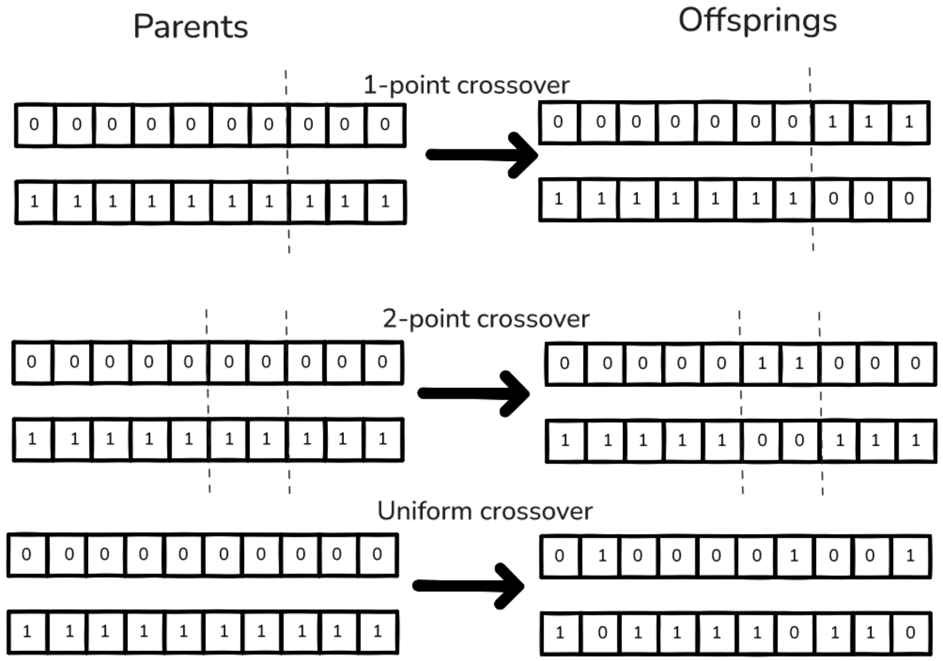
Small exercise 2

- How to define
  - a mutation
  - a crossover

Small exercise 2 SOLUTION

- Mutation is bit flipping

- Crossover, classical crossovers on binary lists works



Small exercise 3

- How look like an EA if an individual represents a permutation of the n first alphabetical letters (for Travelling Salesman Problem for instance)
  - Representation of an individual
  - Proposition of a crossover, mutation operator

A permutation can be defined as a bijection (an invertible mapping, a one-to-one and onto function) from a set S to itself:

$$\sigma : S \xrightarrow{\sim} S.$$

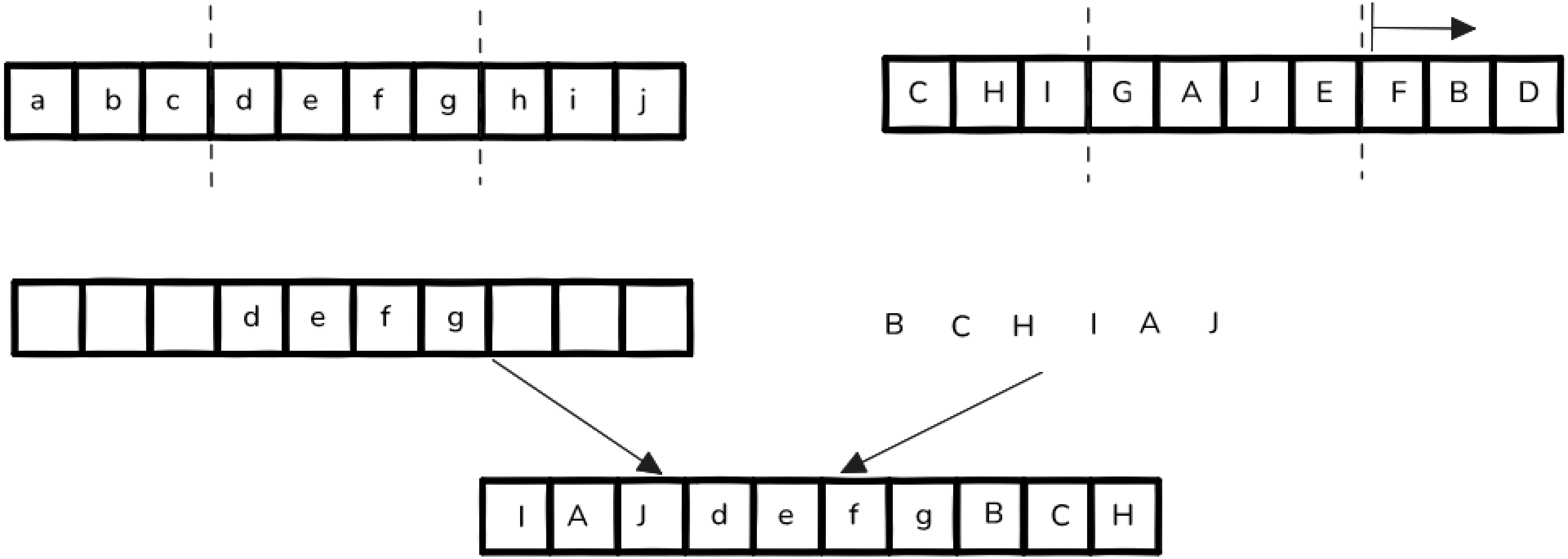
Small exercise 3 SOLUTION

- One solution can be defined by an ordering of the alphabetical list:

$abcdefghij$

- Mutation can be defined by
  - a swap
  - removing an element and place it elsewhere

Small exercise 3 SOLUTION: crossover for permutation



<u>Properties</u>:
From parent 1, the relative order, the adjacency, and the absolute positions are preserved.
From parent 2, only the relative order is preserved.

Outline

- Common concepts on Population based metaheuristics
  - Initial population
  - Stopping criteria
- **Evolutionary algorithms**
- **Swarm intelligence**
  - Ant colonies
  - Particle swarm optimization

What is swarm intelligence

Collective system capable of accomplishing difficult tasks in dynamic and varied environments without any external guidance or control and with no central coordination

- Simple elements that move in the decision space
- Indirect communicate with each other at each generation

Inherent features

- Inherent parallelism
- Stochastic nature
- Adaptivity
- Use of positive feedback (reinforcement learning)

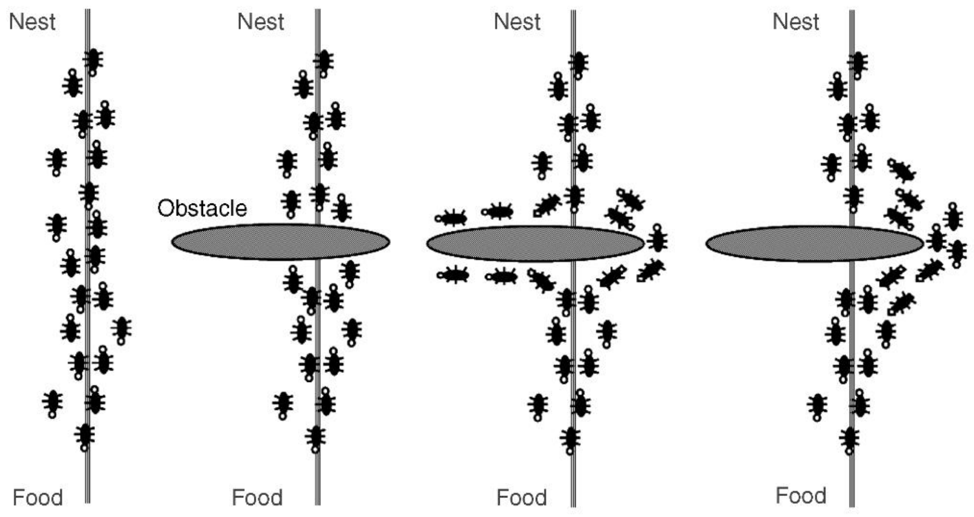
Outline

- Common concepts on Population based metaheuristics
  - Initial population
  - Stopping criteria
- **Evolutionary algorithms**
- **Swarm intelligence**
  - Ant colonies
  - Particle swarm optimization

Ant-colonies

Proposed by Dorigo (1992)

- Imitate the cooperative behavior of ant colonies to solve optimization problems
- Use very simple communication mechanism: pheromone



Ant colonies framework

Algorithm 3.12 Template of the ACO.

```
Initialize the pheromone trails ;
Repeat
    For each ant  Do
        Solution construction using the pheromone trail ;
        Update the pheromone trails:
            Evaporation ;
            Reinforcement ;
Until Stopping criteria
Output: Best solution found or a set of solutions.
```

Definition of ant behavior

A ant travel into a graph, with pheromones being τ_{ij}^{t} for an edge i, j at time t

- At time t, the ant k is at position i have possible next directions N_i^k. The probability to go to a node j is:

$$p_{ij}^{k} = \frac{\tau_{ij}^{t}}{\sum_{l \in N_i^k} \tau_{il}^{t}} \text{ if } j \in N_i^k\text{, else, 0}$$

The next move is made randomly according to these probabilities.

Updates of pheromones

- Initially, a constant amount of pheromone is assigned to all arcs.
- Then:
  - update of the pheromones according to ant behaviors
  - evaporation of the pheromones: $\tau_{ij} = \tau_{ij}(1` - \rho)$

Updates of the pheromones

Multiple strategies, according to the defined problem:

- Online step-by-step: The pheromone trail is updated by an ant at each step of the solution construction
- Off-line: The pheromone train update is applied once all ants generate a complete solution. This is the most popular approach where different strategies can be used
  - e.g, quality based: the (k) best candidates add Λ to all edges traversed $\tau_{ij} = \tau_{ij} + \Lambda$.

Simple Ant-colony construction

- A graph
- A mission to accomplish (e.g., going to one/multiple points)
- A reward according to the path made (duration of the travel)

Initially, a constant amount of pheromone is assigned to all arcs

Main issues in the design

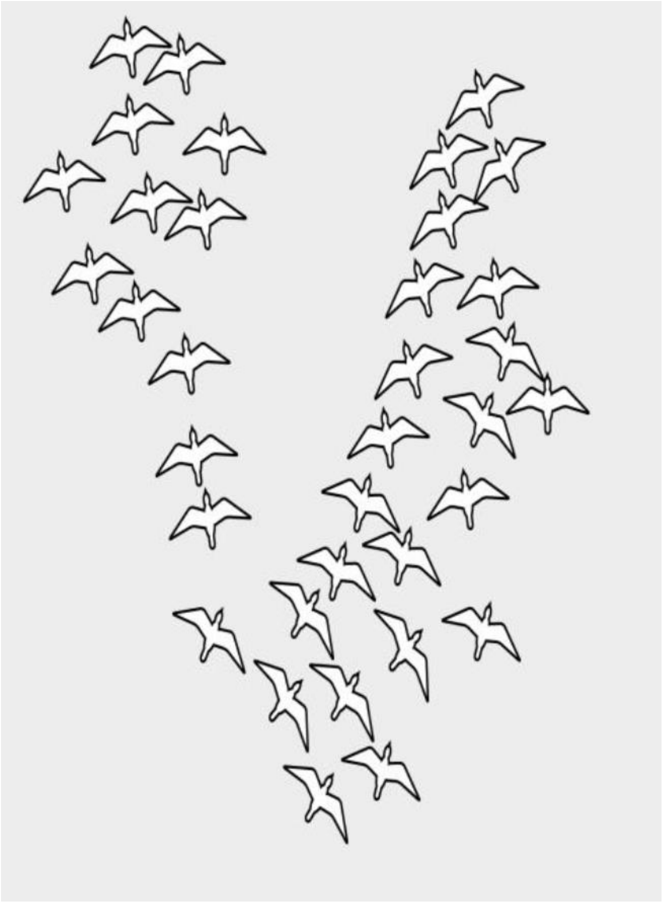
- Pheromone information: should reflect the relevant information in the construction of the solution for a given problem.
- Pheromone update: the reinforcement learning strategy for the pheromone information has to be defined to guide without leading to premature convergence.
- Solution construction: after the run of the algorithm, how to build the solution output. Can be done using a greedy method: the ant that follow each time the path that has the most pheromones.

Outline

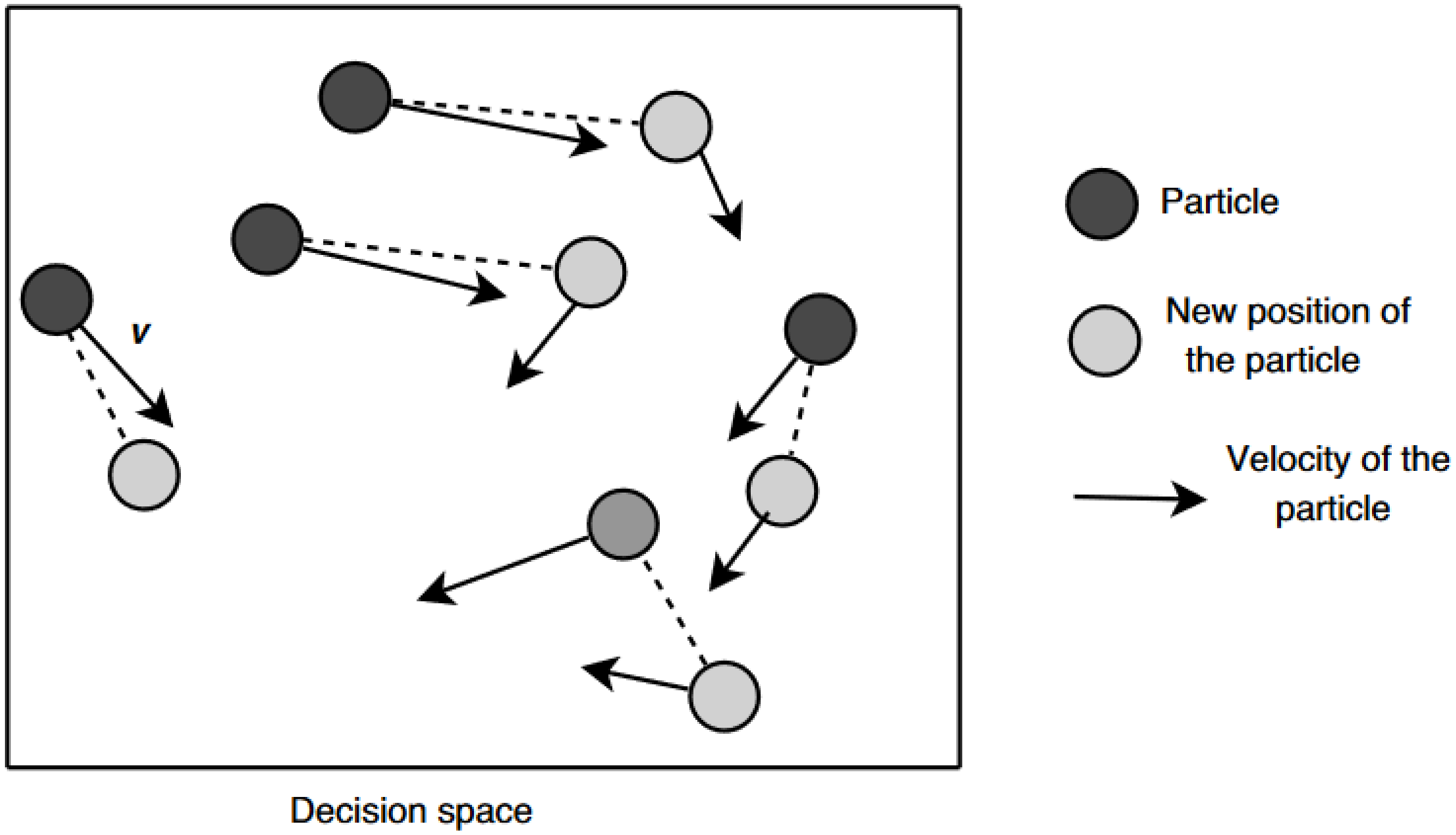
- Common concepts on Population based metaheuristics
  - Initial population
  - Stopping criteria
- **Evolutionary algorithms**
- **Swarm intelligence**
  - Ant colonies
  - Particle swarm optimization

Particle Swarm

- Proposed by Dr. Eberhart and Dr. Kennedy (1995)
- Inspired by social behavior of bird flocking or fish schooling
- Represent an element by its position and veolicity



Particle swarm



Representation of a particle

This problem solve problem that can be represented by a vector of k dimension: analogous to genetic algorithm solution.

A particle is composed of

- The x-vector: current position of the particle $x_i(t-1)$
- The p-vector: best solution found so far by the particle p_i
- The v-vector: a gradient for which particle will travel in if undisturbed $v_i(t-1)$

g-vector represent the position of the best candidate (locally, or globally) p_g

Template of the PSO algorithm

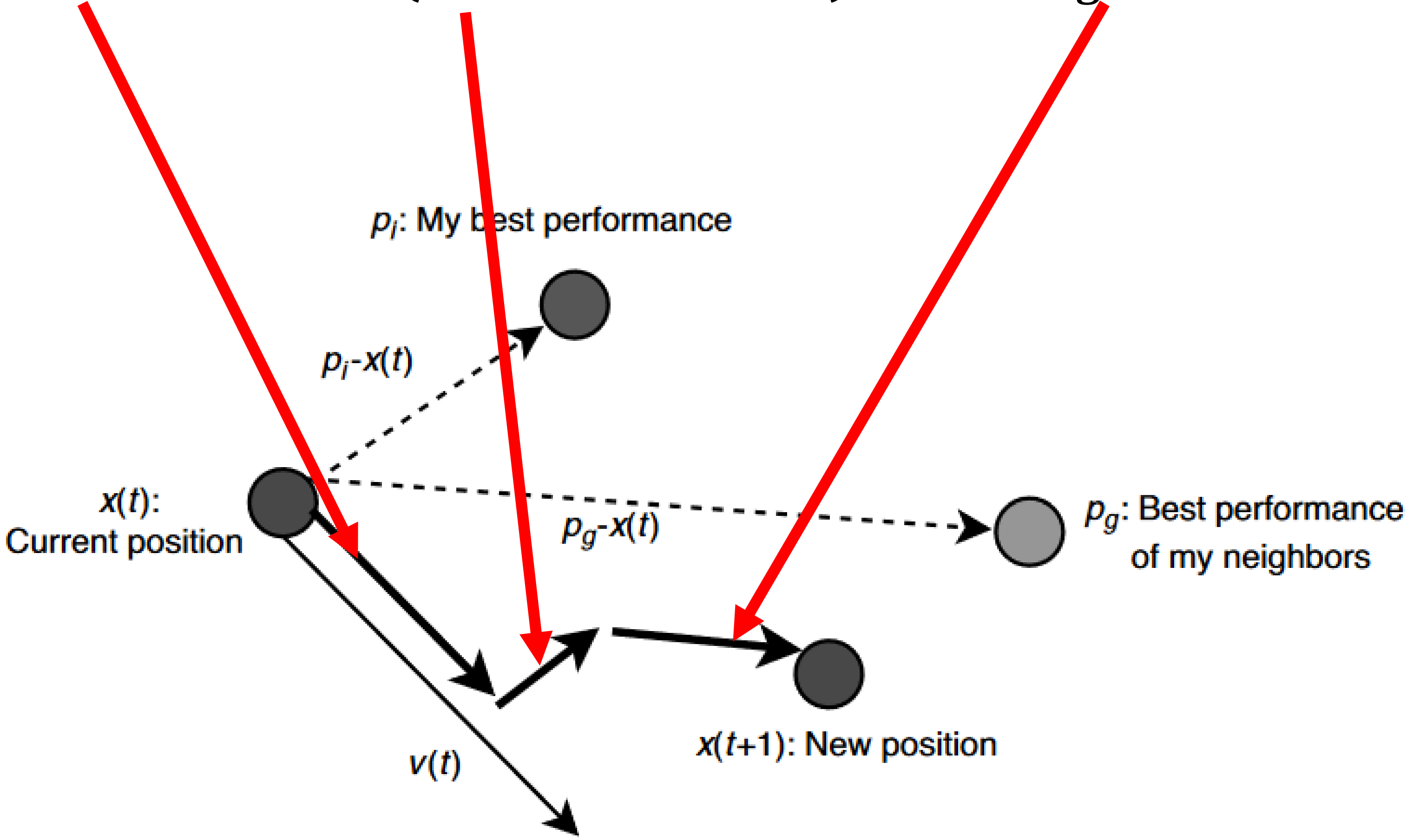
- $v_i(t) = v_i(t-1) + \rho_1\big(p_i - x_i(t-1)\big) + \rho_2(p_g - x_i(t-1))$

Algorithm 3.14 Template of the particle swarm optimization algorithm.

Random initialization of the whole swarm ;
Repeat
  Evaluate $f(x_i)$;
  For all particles i
    Update velocities:
      $v_i(t) = v_i(t-1) + \rho_1 \times (p_i - x_i(t-1)) + \rho_2 \times (p_g - x_i(t-1))$;
    Move to the new position: $x_i(t) = x_i(t-1) + v_i(t)$;
    If $f(x_i) < f(pbest_i)$ **Then** $pbest_i = x_i$;
    If $f(x_i) < f(gbest)$ **Then** $gbest = x_i$;
    Update(x_i, v_i) ;
  EndFor
Until Stopping criteria

Update of a particle

- $v_i(t) = v_i(t-1) + \rho_1\big(p_i - x_i(t-1)\big) + \rho_2(p_g - x_i(t-1))$



Particle swarms: keep in mind

- These methods are more constrained to the structure of the solution in its vanilla phase
- →Can inspired for more advanced methods defined for specific problems

How to build a meta-heuristic: takeaways

- Lots of methods exists, but each depends on:
  - The structure of the solution (binary, list, tree, other?)
  - How to evaluate a solution (costly, explicit..)
  - The link between components of a solution
  - The influence of one good solution to others
- One strategy won't win in all case
  - Not all methods are fitted to a problem (hard to define crossover for instance)
  - Try multiple approaches
  - For one approach, try multiple settings
- These methods are vanilla methods: should work in most cases
  - Adapting the method to a precise problem can improve performance

For the 2 following weeks

- Please bring your computer, as **we are going to work on Python notebook**:
  - Usable through google collab
  - Usable through jupyter (included in the Anaconda package)
  - Included in most python IDEs

University of Luxembourg

Merci | Thank you | Danke

uni.lu
UNIVERSITÉ DU LUXEMBOURG